Express Mail Cert. EL640[illegible]11US
File No. 5035-105 US

14/PRTS

09/720589

526 Re[illegible]PTO 22 DEC 2000

# METHOD OF COPYING A MEDIA FILE

## 1. Field of the Invention

This invention relates to methods for copying continuous media files, such as video and audio files, in which the files stored on the server are encoded to provide enhanced browsing and download features at a device, such as a client or another server. The network may include the Internet.

## BACKGROUND OF THE INVENTION

Recent years have seen considerable interest in improving techniques for delivering all kinds of media files, such as video and audio files, to PC clients over low bandwidth networks, prompted not least by the ever-increasing use of video in Web pages. One useful set of features for video browsing is used in the Indeo product from Intel Corporation. In this system, one encoded video file can be configured by an author to play back at a client at many different data rates and quality levels, each of which is suitable for different network bandwidths or playback-platform capabilities. Using the 'Progressive Quality' option in Indeo breaks down a video download into up to 6 phases, with the first phase delivering the lowest quality frames and subsequent phases delivering higher quality frames. 'Quality' within Indeo can be at any of 3 levels: 'Good', 'Better' and 'Best'. Hence, video quality builds up over time in 3 discrete stages: (i) when the download of data coded for 'Good' quality is finished; (ii) when the download of data coded for 'Better' quality is finished; and (iii) then reaches its highest level when the download of data coded for 'Best' quality is finished.

Reference may also be made to Beong-Jo Kim, Zixiang Xiong and William Pearlman, *"Very Low Bit-Rate Embedded Coding with 3D Set Partitioning in Hierarchical Trees"*, submitted to IEEE Trans. Circuits & Systems for Video Technology, Special Issue on Image and Video Processing for Emerging Interactive Multimedia Services, Sept. 1998. This paper discloses applying a SPIHT compression scheme to a wavelet-based transform, which yields a bitstream encoding multiple spatial resolutions, with progressive quality ordering within a given spatial resolution.

## SUMMARY OF THE INVENTION

In accordance with the present invention, there is provided a method of copying a media file to a device over a network, to enable a copy of the media file to be played, whereby surplus bandwidth that is not required for playing the media file is used to enhance the copy of the media file on the device. Playback and enhancement may be controlled by a set of *playback parameters* and *enhancement parameters* respectively. These sets of parameters are, in one implementation, also known as the *playback profile* and *enhancement profile*.

The surplus bandwidth may be made available as a result of (i) playing the media file at a reduced rate or quality (ii) or re-playing parts of the media file or (iii) pausing playback of the media file. Hence, the present invention contemplates entirely new browsing functionality in which the typical browsing actions of (i) viewing a sequence of video frames at a reduced rate (e.g. a slow motion view) or reduced quality (e.g. fast scan at low resolution) (ii) or re-viewing a sequence of video frames or (iii) pausing to view a single video frame, results in surplus bandwidth being made available; that bandwidth is intelligently used to enhance the copy of the media file at the device so that, for example, the quality of the video frame or sequence of frames (either immediately or subsequently) can be substantially better than at the normal playback rate. This is particularly useful since, when a user browses over a network in these ways, he or she is likely to be interested in scrutinising detail. Enhancement generally occurs as a result of the transmission over the network of refinement data. The present invention has particular application to continuous media files which may be streamed.

The term 'network' should be expansively construed to cover any kind of data connection between 2 or more devices. The network will typically include the Internet. A 'file' is any consistent set of data, so that a 'media file' is a consistent set of data representing one or more media samples, such as frames of video or audio levels. The term 'client' will be used in this specification to mean any device which receives data.

The enhancement may be performed according to a profile that is stored in a server, or transmitted to the server by the device before or during copying, or a combination of values stored in the server and values transmitted to the server by the device before or during copying. Enhancement generally occurs as a result of the transmission over the network of refinement data.

The data required for playback may be selected according to a profile that is stored in the server, is transmitted to the server by the device before or during copying or is a combination of values stored in the server and values transmitted to the server by the device before or during copying.

In one embodiment, the copy of the media file on the device acts as a temporary cache and all or part of the media file is deleted during or at the end of playback. A local cache or a local buffer may be used to store data transmitted using the surplus bandwidth and which is only decoded if required to enhance the copy of the media file on the device.

In one embodiment, the media file encodes video, the image data corresponding to each video frame or sequence of frames is generated using a wavelet transform and modified with SPIHT compression, and the resultant bitstream includes several discrete bitstream layers, each bitstream layer enabling image data at a different spatial resolution to be displayed on a display.

In another embodiment, the device is provided with data for purposes other than playback, including (i) analysis of the media data (ii) re-compression of the media data into a new format (iii) transfer of the media data to another medium such as paper, film, magnetic tape, disc, CD-ROM or other digital storage medium.

Typically, the media file encodes video, audio or combined video and audio.

In one aspect, there is provided a media file which is copied using any of the methods defined above.

In another aspect, there is provided a computer program which enables a client to enhance a copy of a media file copied to it from a server over a network, by using surplus bandwidth that is not required for playing the media file on the client. In a related aspect, there is provided a client programmed with such a computer program.

In a further aspect, there is provided computer program which enables a server to enhance the copy of a media file copied from it to a client over a network, by using surplus bandwidth that is not required for playing the media file on the client. In a final related aspect, there is provided a server programmed with such a computer program.

## BRIEF DESCRIPTION OF THE DRAWINGS

The invention will be described with reference to the accompanying drawings, in which:

Figure 1 is a schematic representation of a network used in performing the method of media file delivery according to the present invention;

Figure 2 is a schematic representing the sub-bands which result from applying wavelet transforms to an image in a conventional multi-scale decomposition with examples of partial reconstruction according to the present invention;

Figure 3 is a schematic representation of the format of the 'chunk' data structure according to the present invention;

Figure 4 is a schematic representing the typical path of a unit of media through the layering system according to the present invention;

Figure 5 is a schematic representation of the labelling mechanism as applied to a wavelet encoding according to the present invention;

Figure 6 is an example of a fragment of a Codebook for the labelling example of Figure 5, according to the present invention;

Figure 7 is a schematic representation of slice filtering as applied to the labelling example of Figure 5 according to the present invention;

Figure 8 is a schematic representation of slice merging as applied to the labelling example of Figure 5 according to the present invention;

Figure 9 is a schematic representation of the labelling mechanism as applied to MPEG encoding according to the present invention;

Figure 10 is an example of a fragment of a Codebook for the labelling example of Figure 9, according to the present invention;

Figure 11 is a schematic representation of slice filtering as applied to the labelling example of Figure 9 according to the present invention.

Figure 12 is a schematic representation of the labelling mechanism as applied to DV encoding according to the present invention;

Figure 13 is an example of a fragment of a Codebook for the labelling example of Figure 12, according to the present invention;

Figure 14 is a schematic representation of slice filtering as applied to the labelling example of Figure 12 according to the present invention.

## DETAILED DESCRIPTION OF THE INVENTION

### A. Key Concepts

**Block-based, motion compensated encoding schemes.**

There are several examples of block-based encoding schemes that use Discrete Cosine Transform, motion detection and compensation to compress video by removing spatial and temporal redundancy from the source. Of these, the most familiar is MPEG (i.e. MPEG-1 or MPEG-2). MPEG utilises three types of compressed picture: I-frames, P-frames and B-frames. An I-frame is built up from 16 x 16 pixel square blocks (Macroblocks) of image, represented as a set of spatial frequencies obtained through the Discrete Cosine Transform (DCT), where the low spatial frequencies are generally represented with much greater accuracy than the high spatial frequencies. Temporal redundancy is removed using P (predicted) and B (Bidirectional) frames. A particular Macroblock in a P-frame is generated at the encoder by searching a previous I-encoded frame (or P-frame) for a Macroblock which best matches that under consideration. When one is found the vector is calculated which represents the offset between the two. This motion vector, together with the error between the predicted block and the actual block, is all that need be sent in order to reconstruct the P-frame at the receiver. The third kind is called a B (Bi-directional) frame. B-frames are based on motion vectors obtained by past and future I and P-frames; they provide a smoothing effect, increase the compression factor and reduce noise.

Because P-frames are computed from previous P-frames, errors can build up, so it is necessary periodically to insert I frames. A typical MPEG sequence of frames may look like this:-

I B B P B B P B B I B B P B B P B B I..........

From the point of view of the present invention, MPEG exemplifies three properties of encoded media files that are factors in the design of a system such as is described here.

The first property is that the initial sample structure is preserved through the encoding, i.e., the identity of individual frames is not lost in the encoded bitstream. This means that temporal properties can be manipulated by adding or removing frames in the compressed domain. Secondly, a temporal window is defined (the MPEG Group of Pictures or GOP) within which temporal redundancy is exploited to achieve compression. Thirdly, a complex set of dependencies is defined by the encoding; in MPEG, P-frames require I-frames, and B-frames require I and P-frames, for decoding.

There are other examples of block-based encoding schemes that utilise motion detection and compensation including H.261 and H.263.

**Block-based intra-frame only encoding schemes.**

There are other schemes, notably JPEG and DV (as defined in SMPTE 314M-1999), that use block-based encoding without motion compensation. In both JPEG and DV, the basic scheme is to transform blocks of pixels into frequency components using the DCT, quantise the components by multiplying by a set of weighting factors, then variable-length code the result to produce the encoded bitstream. DV, however, introduces the concept of *feed-forward* quantisation that optimises the compression process prior to compression being applied. To do this the DCT-transformed image is examined and classified into areas of low, medium and high spatial detail. Using this information, different tables of quantisation factors are selected and used according to area, with the object of matching the fidelity with which frequency coefficients are represented, to the frequency response of the human visual system.

A second feature of DV is it's use of block-based adaptive field/frame processing. What this means is that a 64-entry DCT block can represent either an 8-by-8 area of pixels in a de-interlaced frame (an 8-8 DCT), or two 4-by-8 areas in the first and second fields of a frame (a 2-4-8 DCT). The choice between the two is done by detecting motion. The former scheme is used if there is little motion occurring between fields, the latter if motion is detected; this choice being made on a per-block basis.

As with MPEG encoding, three observations can be made about the nature of the encoded bitstream. As before, the sample structure is preserved through the encoding; secondly, a temporal window is defined which, in this case, represents the two fields of a frame. Thirdly, a set of dependencies is defined: for example, dependencies exist between the fields within a frame wherever 2-4-8 DCT blocks have been generated.

**Subband encoding schemes.**

A more recent alternative to a block-based encoding scheme using a transform such as DCT, is a subband encoding whereby a complete image (rather than small blocks thereof) is processed into a set of frequency/space limited bands, sometimes referred to as ***scales***. An example of this is the Wavelet Transform.

The wavelet transform has only relatively recently matured as a tool for image analysis and compression. Reference may for example be made to Mallat, Stephane G. "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation" IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.11, No.7, pp 674-692 (Jul 1989) in which the Fast Wavelet Transform (FWT) is described. The FWT generates a hierarchy of power-of-two images or ***subbands*** where at each step the spatial sampling frequency - the 'fineness' of detail which is represented - is reduced by a factor of two in x and y. This procedure decorrelates the image samples with the result that most of the energy is compacted into a small number of high-magnitude coefficients within a subband, the rest being mainly zero or low-value, offering considerable opportunity for compression.

Each subband describes the image in terms of a particular combination of spatial/frequency components. This is illustrated in Figure 2A. Here, a wavelet filter is applied twice to an image. After the first application 4 subbands at Level 0 result, with each subband a quarter the scale of the original; after the second application, four new one-eigth-scale subbands are created. To reconstruct an image fully, the procedure is followed in reverse order using an inverse wavelet filter. The Figure also illustrates the scheme used to name subbands: 'L' and 'H' refer to Low-pass and High-pass wavelet filters which are applied in x and y to generate a subband. Applying the H filter in both x and y gives the 'HH' subband, applying 'L' in x and 'H' in y results in a 'LH', and so forth.

Subbands need not always be fully reconstructed into the original image; they may be combined in different ways to produce a final image to satisfy some set of individual requirements. Three examples are shown in Figure 2. In Figure 2B, the LL sub-band of Level 1 is used as a one-sixteenth scale version of the original. In Figure 2C, the four subbands of Level 0 are inversely transformed to reconstruct a one-quarter resolution version of the original. In the third example, Figure 2D, the LL and LH sub-bands of Level 1 are used, together with the LH sub-band of Level 0 to reconstruct an original resolution image, but with horizontal features at all scales emphasised.

**Image Tree-based Compression.**

Tree-based data structures can be used by compression schemes to exploit spatial redundancy in images. A recent development of such a scheme is the SPIHT algorithm; see Beong-Jo Kim, Zixiang Xiong and William Pearlman, *"Very Low Bit-Rate Embedded Coding with 3D Set Partitioning in Hierarchical Trees"*, submitted to

IEEE Trans. Circuits & Systems for Video Technology, Special Issue on Image and Video Processing for Emerging Interactive Multimedia Services, Sept. 1998.

The SPIHT algorithm is an effective compression step for use in conjunction with a wavelet transform, since it can efficiently exploit the decorrelation of the input image provided by the transform to get high levels of data reduction. For the purposes of the present invention a key feature of the SPIHT algorithm is its ability to analyse the transformed image in terms of ***Significance***. It efficiently locates and partially orders all the samples with respect to bit-significance, i.e., the position at which the highest-order bit in a sample is set. Since this corresponds to the magnitude of the sample, the samples are effectively ordered with respect to their energies, or the contribution they make to the reconstructed object. A significance layer is generated by choosing a bit-position and outputting the value of that bit-position (1 or 0) for all the samples for which that bit-position is defined.

As with MPEG encoding, a set of dependencies between related image components is defined by schemes such as Wavelet/SPIHT compression. In this case, a significance layer can be decoded only if the next highest significance layer can also be decoded. Similarly, to decode a subband (e.g. LH0 in Figure 1) the parent subband (LH1 in Figure 1) must also be decoded.

**3D Schemes**

The Wavelet and SPIHT compression schemes can be extended to the third (time) dimension in a fairly straightforward manner. In this case a sequence of frames are captured and are treated as a 3-dimensional block of data. The wavelet filter is applied along the time dimension as many times as it is along the vertical and horizontal dimensions, resulting in a transformed block comprising a set of spatio-temporal cubes, each of which is a subband. The 3D-SPIHT algorithm represent these subbands in terms of octtrees (as opposed to quadtrees in the 2D case) and generates the bitstream accordingly.

Other compression schemes can also be used in the present invention, as will be appreciated by the skilled implementer.

## B. An Overview

### Image Encoding

The method of the present invention requires, in one embodiment, that image data be encoded in such a way that the resulting data for a sequence of frames can be partitioned into two or more slices using an encoding process. Different slices will encode different parts of the sequence of images at different resolutions and quality. It must be possible to decode a sub-set of the compete data set in order to re-construct the sequence of frames at a particular displayed resolution, quality and frame rate.

By way of preferred example, partitioning of a bitstream generated using a wavelet transform and SPIHT compression into slices is described later on in this specification. By decoding one or more of these slices of data, it is possible to re-construct the frame or sequence of frames at a particular displayed resolution, quality and frame rate. In general, the quality of the resulting frame or sequence of frames will depend on the total amount of data that is being decoded. The choice of slices that are used to re-construct the frame or sequence of frames can be made on the basis of the desired resolution, quality and frame rate. Also, it is possible to improve the quality, resolution and frame rate of an existing frame or sequence of frames by adding more slices. This general process is known as "enhancement". (In the wavelet-based example given later, we also use the term 'refinement' for this process.) Typically, enhancement occurs by adding new slices to existing layers or by adding the first slices from a new layer; possibly, but rarely, it may involve adding an entire new layer (i.e. all the slices for that layer).

## C. System description

The system capable of performing the method of the present invention comprises an Encoder, a Network, a Server and N clients, as shown in Figure 1.

A Encoder compresses the incoming media, structures it into layers and adds layer labelling information as described in the next section, prior to transmitting it as a bitstream over a digital communications network, to be stored on the Server.

In Figure 1, there are N clients, each of which engages in a session with the media Server during which media content and control information are transmitted. For each client a control channel to the Server is maintained which allows that client to request that media be transmitted to it. The types of requests that can be made include (but are not limited to) the following:-

- Seek to a particular point in the media file.
- Transmit a media clip at a particular quality.
- Transmit extra information to improve in a specified way the quality of the media at the Client.

These requests support a variety of useful application-level functions at the Client such as video previewing using fast-forward and rewind, single-frame stepping forwards and backwards, and freeze-frame. For example, in a media browsing application, a low-quality version of the media may be sent to the client with very little latency, to allow the user to start work immediately, for example, on rapidly scanning through a file to locate items of interest. Initially the user may not so much be interested in the absolute quality of the media, as in the responsiveness of the system to requests to move rapidly from one point to another. Having found an item of interest the application may then determine that this particular section of the media is to be rendered at greater quality and issue an appropriate request to the Server. The Server calculates which extra layers should be delivered to improve the media information in the way required and transmits only those layers. Note that no extra processing of the media file is required during this procedure other than to locate and send the appropriate sets of data. Note also that the quality improvement may occur in a number of ways, including:

- Increasing the temporal resolution of the media (i.e., for video, allow more individual frames to be resolved).
- Increasing the spatial resolution of the video.
- Increasing the sampling frequency of the audio.
- Decreasing the degree of distortion of the media.

Central to the working of this scheme is the mechanism used to convert a media file in any encoding format, including those described in the preceding section, into a layered format with properties that support the operations described above. This is the purpose of ***Layer Labelling*** as described in the next section.

## D. Description of Layer Labelling

### Overview

A Layered bitstream is built out of a media encoding by splitting it up into packets (called ***chunks*** or slices here) in a manner determined by the encoding type, and appending ***Labels*** to the chunks that uniquely describe their contribution to the reconstructed file. In order that distributed tools can be built and configured to understand the formats of many layered bitstreams in a distributed environment, ***static*** signalling information (constant throughout a stream's lifetime) is made available, either as a reference to a globally available file, or as data sent once in the bitstream. For configuration information that must vary within the stream, provision is made for ***dynamic*** signalling information to be carried in the bitstream.

To be successful the labelling scheme must do three things well:

- Efficiently and controllably allow parts of a media file to be selected to obtain a particular quality of service to satisfy media quality, bit-rate or latency requirements.

- Efficiently support the location and transmission of those parts of the media necessary to reconstruct a particular item to a specified quality.

- Encapsulate the underlying media encoding and hide complex details while offering a uniform, media encoding-independent, layered view of the file.

The first requirement is addressed through the concept of a ***Filter*** that specifies which parts of the file are to be selected for transmission. The second requirement needs in-band ***signalling*** information to describe and delineate the layered structure within the bitstream. The third requirement is addressed through an indirection mechanism that maps the subtleties of a particular encoding to the uniform layered view. This indirection mechanism is termed the ***Codebook***. A Codebook is defined for every new encoding scheme, or variations of existing schemes with new parameters.

## Layer Labelling Format

This section defines the format of the layer labels used in the present invention to convert a media file to a layered stream format. Streams are stored and transmitted as contiguous groups of chunks, each with a chunk header defining the length and type of the chunk. Some chunk types are reserved for carrying data (termed slices), whereas others are used for signalling.

Extra configuration information for these streams, such as encoding format and slice dependencies, can be associated with the data stream. The static portion of this information can be stored externally, or in-band using signalling chunks; dynamic information is always stored in-band.

## Chunk Header

All chunks are introduced by a chunk header, the format of which is illustrated in Figure 3. All chunks, irrespective of their type, are introduced by one of these headers. The result is a data stream whose structure is fixed, while still allowing room for expansion in the future.

### *Type*

The 2-bit type field is used to distinguish between 4 independent sets of chunk types. (The issue of independence is important, since there is no way to specify a dependency between two labels belonging to different types.) The following type values are currently defined:

*data chunks (slices)*

*signalling chunks*

### *Length*

The 22-bit length field gives the length of the chunk in bytes, including the header. A chunk containing no payload information (i.e. just a header) would, therefore, have a length of 4.

***Label***

The 8-bit label field is the layer label for the chunk. The interpretation of the label is dependent on the type field (see above), so the header format allows for 4 sets of 256 labels.

**Terminology**

Data chunks (equivalently termed *slices*) are used to encapsulate a stream of multimedia data. The labels attached to the slices are used to indicate the nature of the slice contents and their dependency relationships with other slices in the stream. Signalling chunks can be used within the stream to pass extra information about the data, such as details about how it has been encoded or filtered.

***Slice***

The manner in which the slice label allocation is done depends on the exact nature of the multimedia data being encapsulated. There are a set of basic schemes which are mutually agreed between all the tools, but new allocation schemes can be added in order to tailor the system to suit different situations. The only restrictions are:

- slices within a data stream must all be labelled using the same allocation scheme;
- slices must be labelled uniquely with respect to the data they contain; and
- label allocation must be done in slice dependency order.

The first restriction is self explanatory, but the others merit further discussion. The slice label is taken as an indication of the data contents of the slice, and is used by the system to separate and merge the slices to obtain data streams of differing quality. In order to allow the stream decoders to be stateless, the slice label must identify uniquely the actual data contents. As a result, slices which are too long to be encapsulated as a single chunk must be split in such a way as to enable the decoder to determine which is the base data and which is the continuation information.

The slice labelling must also reflect the dependency hierarchy, with all slices having numerically lower label values than those which depend on them. This means that when the slices of a stream are encoded in dependency order, the label values increase in line with the progress through a dependency unit (see *context* below). Not only does this simplify the job of merging the slices from data streams which have been split previously, but also allows the boundaries between contexts to be implicit (if a slice label is not numerically greater than its predecessor, then it must belong to a new context).

***Context***

A group of slices which are related by dependency (either spatial or temporal) is called a context. As described above, the label values for the slices within the context must be arranged so that they are in numerically increasing order. This usually allows the boundaries between contexts to be detected implicitly, without the need for an explicit signalling chunk. Based on the label allocation scheme described above, slices within the context will also, therefore, be arranged in dependency order such that slices always precede those which are dependent on them.

Because of the number of slice labels available (and the restriction of unique slice labelling), it follows that the maximum number of slices which comprise a context is 256. The dependency hierarchy for the slice labels is defined in a code book (see below) and is fixed for the duration of the data stream. Note that it is not mandatory for slice labels to be allocated consecutively (i.e. gaps in the allocation scheme are allowed); nor is it essential for a specific slice label to be present in all contexts.

As well as grouping together slices which are related by dependency, contexts have a very important rôle in defining the boundaries at which editing can be performed. There is no need for explicit signalling chunks to determine valid edit points.

Contexts are assumed to be separated from their neighbours in the temporal domain, and be independent of them; in other words, when contexts are extracted from the data stream, each should be capable of being decoded as a stand alone entity. The one exception to this is exemplified by encoding schemes such as MPEG which have temporal dependencies between adjacent Groups Of Pictures. This is handled by associating an "overlap" with each context allowing temporal units (see sample below) to be marked as dependent on the immediately preceding context.

The overlap value, stored in the media header (see below), defines how these inter-context dependencies are handled: an overlap value of n implies that the first n samples of a context are dependent on the previous context. Since the normal situation is for contexts to be temporally self-contained, the default overlap value is zero. In the case where it is not zero, the slice dependency hierarchy must reflect the inter-context peculiarities of the encoding scheme, with "phantom" dependencies being added if necessary.

***Sample***

Multimedia data streams are usually represented as a sequence of discrete units, each with a well defined temporal position in the stream. This is particularly true with video data, which is usually modelled as a set of separate frames (even though the frames may have been reordered and grouped in the encoding process, such as with MPEG). The data streams preserve the natural temporal unit of the encoding scheme, with each discrete unit being termed a sample. Whether a context contains a single sample or a group of samples is dependent on the encoding technique used, but specific techniques usually follow a very rigid repeating pattern. By default, therefore, each context is assumed to contain a fixed number of samples, with the context spacing (samples per context repeat count) defined in the system header (see below).

The distinction between contexts and samples is important when contemplating the temporal dependencies of the multimedia data (such as a 3-D

Wavelet/SPIHT scheme) and the ability to perform temporal decimation (such as playing every fourth frame of a video sequence). A context contains the smallest number of samples which make up a temporal encoding unit (GOP for MPEG, GOF for Wavelet/SPIHT), with the spatial and temporal dependencies being handled in exactly the same manner. Within the context, each slice is given a temporal priority (see below) which allows for intelligent decimation of the temporal sequence, but does not in itself imply any kind of temporal dependency.

***System Header***

The system header is used to define the stream attributes which are relevant at the system layer only. Basically, this means parameters which are needed by the Server to provide the level of service required of them. For example, the default context spacing is needed in order to construct the mappings between contexts and samples.

***Media Header***

The media header is used to define the stream attributes which are relevant at the media layer only. This means parameters which are needed by the stream decoder in order to make sense of the data stream, but which are not required by the Server. Examples which fall into this category are: the horizontal and vertical resolutions of a video encoding; the inter-context dependency overlap; and a reference to the code book which has been used for the encoding.

The reason there is a separate media header chunk, rather than combining the information with that in the code book is because code books are generic in nature and tend to be independent of the raw properties of the original media. The media header fills in these details for a specific recording, thus greatly reducing the number of code books which are required by the system as a whole.

***Code Book***

The code book is used to define the dependency and quality relationships of the slices within a context, and is necessary in order to construct a filter for a data stream. Irrespective of whether the dependencies are spatial or temporal, the relationships between the individual slices can be represented as a simple hierarchy: each slice type (i.e. slice with a specific label) has a predetermined set of other slice types on which it is dependent. The nature of the hierarchy is, obviously, dependent on the labelling scheme used; but the code book technique allows a great deal of flexibility providing the basic rules of slice labelling are not violated (see above).

In the context of the code book, "dependency" has the following meaning. For one slice to be dependent on another, it must be impossible to decode the slice without it. (This is true, for example, with MPEG video where the I frame is necessary before the first P frame can be decoded.) Note that this dependency relationship says nothing about the desirability of slices when it comes to producing results of acceptable quality.

As well as defining the dependency hierarchy, it is the job of the code book to store a mapping between the individual slice values and the "quality" of the decoded data stream which include those slices. The stream quality is, by necessity, a vague and subjective factor compared to the strict slice dependencies described above; indeed it is likely that there is more than one quality dimension. For example, 3-D Wavelet/SPIHT encoded video can be thought of as having four quality axes: scale (image size), fidelity (image quality), colour and temporal blur.

The bulk of the code book comprises a series of tables (one for each slice label) containing the dependency data, the temporal priority and a set of quality parameters, one for each quality axis. The number and type of quality axes defined is dependent on the nature of the multimedia data and the encoding scheme used. To simplify matters, three important restrictions are applied to quality axes:

Quality axes are assigned names (or quality tags) which have global significance. Code books which employ the same quality tag must use the same allocation scheme when it comes to quality parameters for the relevant axis.

Quality parameters are always specified with regard to the highest quality (unfiltered) version of the data stream. In the case of video, for example, this allows a common code book to be used irrespective of the image size.

The quality parameters themselves must, where practical, be independent of the actual encoding scheme used. For example, the scale parameters for video data might represent the width or height of a scaled image compared to the original, assuming an unchanged aspect ratio.

The code book header contains the list of quality tags for which quality parameters will be found in the tables which follow. This, combined with the restrictions outlined above, allows for filter creation to be done in a manner which is entirely independent of the multimedia data type and encoding scheme. A filter created using a quality axis which is either absent or unspecified always results in slices being unfiltered with respect to that axis.

***Temporal Priority***

It is often necessary to decimate a data stream, either to reduce the bandwidth requirements or to play it at faster than normal rate. Given the different data encoding schemes, it is not always sensible to work with naïve sub-sampling schemes such as "one sample out of every four". There are two reasons for this.

The inter-slice dependencies of the relevant samples may prohibit simple decimation. This is true, for example, with MPEG video, where there is a complex web of temporal dependencies between the I, P and B frames.

When attempting to maintain a cache of slice data and ensure effective use of available bandwidth, it is essential that repeated (or refined) sub-sampling requests should be carefully orchestrated so as to maximise the data overlap.

Accordingly, each slice in a context is assigned a temporal priority which is used to control decimation. For example, in MPEG video, slices belonging to the I, P and B frames would be allocated temporal priorities of 0, 1 and 2 respectively. It is these temporal priorities which are used when performing temporal filtering below the level of the context.

**Signalling Chunks**

The signalling chunks are used to identify and annotate the data chunks in a data stream. Whether they are stored together by Servers or generated on the fly at the time of delivery is an implementation detail which is outside the scope of this document. The signalling chunks fall into three distinct categories: static, dynamic and padding.

***Static chunks***

The static chunks define parameters which do not change during the lifetime of the data stream. This includes the definitions of the basic default values which apply to the whole stream, such as the identity of the code book which has been employed to generate the data slices. Since these chunks are static, there is no necessity for them to be included as part of the data stream: it is just as valid for the information to be sent out-of-band or, as in the case of re-usable code book tables, by reference. If they are transmitted or stored in-band, however, they must be unique, precede the first data slice and be stored in numerically increasing opcode order.

***Dynamic chunks***

The dynamic chunks define those parameters which vary depending on their position within the data stream. This includes filtration information, since the filters used to generate the stream can vary as the slices are delivered. It also includes all the variable context and sample information, such as the indication of "seek" operations and the handing of contexts with a context spacing which is not constant. Dynamic chunks, by their very nature, carry positional information within the data stream, and hence must always be sent in-band. Where present, dynamic chunks are only valid at context boundaries.

***Padding chunks***

The padding chunks have no semantic influence on the data stream, and hence can be positioned at any chunk boundary.

**Static Signalling Chunks**

**<u>System Header</u>**

**Type:** Static

**Opcode:** 0x00

**Contents:** Information which is specific to the <u>system</u> layer only (Media Server), and is not required to filter or decode the data stream. The parameters are:

*context spacing*

**Format:** Name/Value pairs.

**Status:** Mandatory.

**Position:** Out-of-band, or in-band before first data slice.

**<u>Media Header</u>**

**Type:** Static

**Opcode:** 0x01

**Contents:** Information which is specific to the <u>media</u> layer only (Decoder), and is not required for the operation of the Media Server. The parameters are:

*original media parameters*

*context dependency overlap*

*code book reference*

**Format:** Name/Value pairs.

**Status:** Mandatory.

**Position:** Out-of-band, or in-band before first data slice.

**Code Book**

**Type:** Static

**Opcode:** 0x02

**Contents:** Information which is specific to the generation of slice filters only, is not required for the operation of the Media Server or Decoder.

*quality tag list*

*slice dependency information*

*slice quality information*

*slice temporal priority information*

**Status:** Mandatory.

**Position:** Out-of-band, by reference, or in-band before first data slice.

**Meta Data**

**Type:** Static

**Opcode:** 0x04

**Contents:** Information which is pertinent to the data stream, but which is not required for its storage, filtration or transmission. Examples are:

*date, time & location*

*edit history*

*copyright notice*

**Format:** Name/Value pairs.

**Status:** Optional.

**Position:** Out-of-band, or in-band before first data slice.

<u>**User Data (1,2,3,4)**</u>

**Type:** Static

**Opcode:** 0x05, 0x06, 0x07, 0x08

**Contents:** Private information, added into the data stream by the generator of the encoding, but which is not interpreted by the system in any way.

**Format:** Opaque byte array.

**Status:** Optional.

**Position:** Out-of-band, or in-band before first data slice..

**Dynamic Signalling Chunks**

<u>**Baseline**</u>

**Type:** Dynamic

**Opcode:** 0x80

**Contents:** Information regarding the stream's current position with respect to the context and sample sequence numbers, and is usually used to indicate a "seek" operation within the data stream. Has two parameters, either of which can be omitted if the value can be inferred from the stream position:

*context sequence*

*sample sequence*

At the start of a data stream, unless otherwise specified, it is assumed that the context and sample sequence numbers both start at zero.

**Format:** 64-bit little-endian binary.

**Status:** Optional.

**Position:** In-band at context boundaries.

**Context Start**

**Type:** Dynamic

**Opcode:** 0x81

**Contents:** A *context spacing* for the immediately following context.

**Format:** 32-bit little-endian binary.

**Status:** Optional. Only necessary if the *context spacing* is different to the default.

**Position:** In-band at context boundaries.

**Context End**

**Type:** Dynamic

**Opcode:** 0x82

**Contents:** None.

**Format:** n/a

**Status:** Optional. Only necessary if context boundary cannot be determined by simple comparison of slice type values.

**Position:** In-band at context boundaries.

**Filter Definition**

**Type:** Dynamic

**Opcode:** 0x83

**Contents:** Used to encode stream filtration information for the data slices which follow. It takes the form of a pair of bit sequences (possibly compressed) indicating which data has been filtered out of the subsequent stream.

The *slice mask* indicates which of the 256 possible slice types have been filtered out of the data stream. It does not imply that the remaining slice types will be found in the stream; nor does it guarantee that the slices which are present conform to the dependency criteria specified in the code book.

The *context mask* is used to indicate which whole contexts have been filtered out of the data stream. Each context has a unique position in the sequence, and mask refers to the position value modulo 240 (a number chosen because of its numerical properties, being the common multiple of 2, 3, 4, 5, 6, 8, 10, 12, 15, 16, 20, 24 and 30). Unlike the slice mask, the context mask can be used to determine which contexts are present and which are absent. In the case of inter-context dependencies, however, there is no guarantee that the relevant information is actually present.

**Status:** Optional.

**Position:** In-band at context boundaries

## Padding Signalling Chunk

**Padding**

**Type:** Dynamic

**Opcode:** 0xff

**Contents:** Opaque byte array. Padding chunks may be necessary in the case where fixed bit-rate streams are encoded or specific byte alignments are required. They can also be used to re-label a slice in place for data filtration experiments.

**Format:** Opaque byte array.

**Status:** Optional.

**Position:** In-band.

## Layer Labelling, Filtering, Merging and Codebook Examples.

Figure 4 gives an overview of some typical transformations undergone by a media file as it is layered, filtered and decoded. In particular, the Figure shows the two styles of filtering that are possible: filtering at the Slice level (within a dependency unit), and filtering at the Context level (across entire dependency units). Slice-level filtering can be used to change the colour depth, spatial resolution, fidelity or temporal resolution of the media. Context-level filtering has a coarser 'grain' and is designed mainly to support temporal decimation of the media.

**Wavelet Example**

Figure 5 shows a more detailed example of labelling Wavelet/SPIHT-encoded data. A depth 2 wavelet decomposition with four significance levels results in a Context of 28 slices. A fragment of one possible Codebook for this encoding is shown in Figure 6 and an example of how the Codebook is used to generate a Filter is illustrated in Figure 7.

Referring to Figure 6, the header of the Codebook includes a **CodebookID** field for self-identification, a **QualityTags** field that specifies the quality axes available for this encoding, a **QualityDivisions** field that specifies, using plain-text names, how those axes are labelled, and an **OriginalSize** field that specifies the spatial resolution of the original media, to provide a baseline for scale calculations.

Each encoding scheme may have its own individual Codebook defined, but any Codebooks that share common **QualityTags** entries must use exactly the same **QualityDivisions** scheme to label the quality axes. Textual names are used to map properties of the different media encodings, which may be similar but not identical, into a single quality scheme. This results in an important capability of the system: that an application can be written to manipulate media using these names in a manner that is completely independent of encoding format.

In the Figure it is assumed that a file manipulation tool wishes to extract a medium fidelity, monochrome, half-scale version of a 352 by 288 video encoding. To accomplish this the Codebook is searched (in practice this is optimised using pre-calculated indices) for the highest-numbered label with the required value of

**QualityParams**, i.e., "scale=half, fidelity=medium". Having found this label (13) all the dependent labels are visited until the dependency graph is completely resolved. As this proceeds the bit mask representing the Slice Filter is built: a 'zero' at bit position *n* indicates that the slice with label (*n*) is required in the filtered bitstream, a 'one' means that it should be absent. The result of using this filter is illustrated in Figure 7.

Filtered slices are preceded by a Filter Definition signalling chunk which holds the slice mask used to perform the filtering. This value can be used by downstream tools that require knowledge of the stream characteristics, for example, to initialise a decoder, or set up memory areas for buffering. The Slice Mask is also used when filtered bitstreams are merged to determine the slice content of the resulting stream, as illustrated in Figure 8. Here, a refinement stream containing slices 16, 17, 20, 21, 24 and 25 is merged with the half-scale, low-resolution filtered stream of Figure 7 to obtain a full-scale, low-resolution encoding. The slice mask for the combined stream is obtained using a simple bitwise OR of the two input Slice Masks.

**MPEG Example**

Figure 9 shows an example of labelling MPEG-encoded data, and Figure 10 shows a fragment of the corresponding Codebook. In this example it is assumed that a file manipulation tool wants to generate a Filter that operates on slices in such a way as to generate a fast playback stream. In this encoding a Slice represents an MPEG frame, so the effect of the Filter is to decimate the stream temporally. From the Codebook the tool resolves the dependence hierarchy for the QualityParams playrate=fast and builds the Slice Filter as described above. The result of applying the Filter is shown in Figure 11.

**DV Example**

Figure 12 shows an example of labelling DV-encoded data, and Figure 13 shows a fragment of the corresponding Codebook. In this example the DV-encoded data is labelled according to the following scheme: data that represents both 2-by-4-by-8 and 8-by-8 DCT blocks from field 1 are assigned slice label 0; data that represents 8-by-8 DCT blocks only from field 2, are assigned slice label 1. This results in two layers, a base layer with half-vertical resolution derived from field 1, and the refinement layer that produces the full-resolution reconstruction. Figure 14 illustrates the result of applying a Filter that selects label-0 Slices only, so producing a half vertical resolution video stream.

## E. Client-driven image delivery

The method described in this specification allows a client to specify information about an image that it wishes to receive and have the server deliver the appropriate slices of the image data at the appropriate time. The client can specify (amongst other parameters) the image size, the image quality, the frame-rate and the data rate. The server will use this information to decide which sections of the image data should be sent to the client and at what rate.

The client can also specify a playback rate at which the data is delivered. The display time of each frame is used to deliver the data in a timely manner appropriate for the selected rate.

## F. Playback parameters

When a client requests the delivery of data it specifies a set of *playback parameters* that are typically specific to the needs and circumstances of that client. The playback parameters may specify the allowable value for any quality parameters that are specified for the media. The playback parameters may be specified by the client; they may already be stored on the server or they may consist of a combination of client and server parameters.

In particular the playback parameters may define one or more of the following media parameters for an image or sequence of images:

(i) the spatial resolution of the image or images;

(ii) the level of distortion in the image or images;

(iii) the number of displayable images;

(iv) the selection of colour components within one or more images;

(v) a sub-set of the available frames to be delivered; and

(vi) a region of interest within one or more frames;

The playback parameters may also define one or more of the following media parameters for audio:

(i) the distortion of the audio;

(ii) the dynamic range of the audio;

(iii) the number of audio channels to be transmitted; and

(iv) selection of monophonic, stereophonic or quadraphonic audio.

In the Codebook case some of these parameters may be encapsulated in a client-generated or server-generated filter for the media.

**G. Server playback**

The enhancement may be specified by the client; they may already be stored on the server or they may consist of a combination of client and server parameters. The server uses the playback parameters to deliver the appropriate data to the client. This process involves the following steps:

1) The server determines the total set of slices to be transmitted.

2) The server delivers the slices at the appropriate rate.

**H. Determining the set of slices for playback**

The server selects the slices that need to be transmitted on the basis of some of the playback parameters. It starts with a complete set of slices and then discards slices on the following basis:

- It discards all slices that code colour components that are not required.
- It discards all slices that code only for frames that lie outside the specified set of frames.
- It discards all slices that code only for frames that are not required for the selected frame rate.
- It discards all slices that code for a quality greater than the required quality that have parent layers that code for at least the required quality.

- It discards all slices that code for a scale greater than the required scale that have parent layers that code for at least the required scale.
- It discards all slices that are marked as already available on the client.
- It discards all slices for which any quality parameter lies outside the allowable range specified in the playback parameters for that parameter.

Other selection criteria may also be applied at this point.

**I. Delivery of the slices for playback**

The server delivers the selected slices to the client at the data rate required for display at the specified playback rate. It uses the timing information associated with the slices to deliver the data in accordance with standard pacing mechanisms for streaming time-dependent media.

**J. Discarding data sections**

In order to display the resulting frames in a timely manner the data must be delivered at the requested rate. When this is not possible the server must discard some of the slices. This can happen because the data rate of the selected slices is higher than the maximum data rate specified in the playback parameters, or it can happen because the transmission channel does not have sufficient capacity for the selected data rate on a permanent or temporary basis. Data is discarded according to a list of criteria, including those contained in a sub-set of the playback parameters known as the discard parameters. The most important criterion is the dependence between data sections and their parents. The parents of a data section are never discarded before the data section itself. The remaining criteria and the order in which they are applied are specified by the client in the discard parameters. The following criteria can be applied:

- Slices that are required for a given frame-rate or higher may be discarded
- Slices that code for a given scale or greater can be discarded
- Slices that code for a given quality or better can be discarded
- Slices that code for a specific region can be discarded
- Slices that code for specific colour components can be discarded

- Slices with quality parameters that exceed a given limit value may be discarded

Other criteria may also be applied at this point. Each criterion in the list may refer to any of the quality parameters that apply to the slices, and the same quality parameter may appear in multiple criteria. For example, the discard parameters may first discard all data above a certain quality, then discard alternate frames and then discard all data above a second, lower quality. Data is discarded until the required data rate is at or below the available capacity at the time.

**K. Changes to playback parameters**

The client may change the playback parameters at any time. The server will re-calculate the new set of required slices, allowing for the slices that have already been delivered to the client. If a slice is currently being transmitted, transmission may be completed unless the slice no longer appears in the required list, in which case it may be terminated. If the transmission is terminated the data stream is marked accordingly to notify the client of the incomplete slice.

**L. Data storage by the client**

When the client successfully receives the data it is stored for decoding and re-use. At a later time the server can enhance the stored data by transmitting other slices for the frame or sequences of frames. The client will add this data to the existing stored data to give compressed data that can be decoded to generate a frame or sequence of frames at a higher quality, resolution or frame-rate. When the time comes to display a frame or sequence of frames, the client can use a combination of stored data and data that is currently being delivered from the server to provide the compressed form of the frame or sequences of frames. This data is then decoded and displayed as appropriate.

**M. Dealing with limited channel capacity**

The data channel will have limited capacity depending on factors such as the overall capacity of the network, the network usage and the capacity that is allocated to the channel. If the capacity requested by the playback profile is greater than the available capacity, the server will reduce the amount of data that is sent using the criteria given by the discard parameters. If the capacity requested by the quality profile is less than the capacity available on the channel, or if it is not possible to exactly match the available data rate for playback, there is said to be *surplus capacity.*

**N. Surplus channel capacity**

Surplus capacity can become available in a number of ways. For example, the client may specify a maximum quality setting in the playback profile while altering the required frame-rate depending on user action. When the video is being played in slow motion, the capacity required for the video data may be reduced, leading to surplus capacity. In the extreme case the user may halt playback in order to display a still image. Once the still has been transferred at the required quality, no further data will be required and the entire channel capacity will be available as surplus capacity. Likewise the client may re-play a frame or sequence of frames that has previously been delivered. Some of the required data for the frames will already be stored by the client so that less capacity will be required for playback of the data at the specified quality. The playback parameters may also be chosen to specify a quality that can be delivered without using all the available channel capacity. The remaining capacity is surplus capacity. Thus some or all of the channel capacity may be surplus.

**Using Surplus Capacity**

When there is surplus capacity on the channel it can be used to improve the quality of video frames by pre-loading some or all of the image data for those frames. This may involve loading data for new frames that have not been previously downloaded (i.e. 'supplementing' previously downloaded frames) or it may involve enhancement of existing frames by downloading more slices for those frames. This process is known as *enhancement* and is under the control of a set of *enhancement parameters.* 'Enhancement' therefore also includes supplementing previously downloaded frames. The set of enhancement parameters is also known as an *enhancement profile*, and it is used to select the order in which the frames that are to be pre-loaded and the amount of data that is to be loaded for each frame.

For example, surplus capacity may be used to load a low quality version of an entire sequence to allow subsequent fast play through the material. Or the surplus capacity may be used to load certain *key frames* for the sequence for scrubbing through the material. Or it may be used to improve the quality of the frames surrounding the current playback position.

The data may be retained on the client for re-use at a later time, or it may be treated as temporary and discarded once the client has no more immediate use for it.

In some cases there may be problems delivering data from the server to the client. For example, delivery might be over a network using an unreliable transport protocol. In this case there may be gaps in the stored data due to loss of data on the network. The surplus capacity can be used to supplement or fill-in these gaps to provide a complete set of compressed data.

## O. Server enhancement

The server uses the enhancement parameters to deliver the appropriate data to the client. This process involves the following steps:

- The server determines the total set of slices to be transmitted.
- The server orders the slices in accordance with the enhancement parameters.
- The server delivers the slices using all the available surplus capacity.

### Selecting slices for enhancement

The enhancement parameters are used to select the slices for enhancement in the same way that the playback parameters are used to select slices for playback.

### Ordering slices for enhancement

Once the enhancement slices are selected there is then an additional step in which the slices are re-ordered to match the requirements of the enhancement parameters. This allows the client to specify the order in which the slices are delivered and hence the way in which the surplus bandwidth is used to enhance or supplement the local copy of the image.

Slices are ordered by assigning a score to each slice. This score is calculated on the basis of the value for each available quality parameter and the importance assigned to that quality parameter in the enhancement parameters.

The following parameters can be used to score a slice:

- The scale/resolution available within the given slice.
- The distortion present within the given slice.
- The frames that are encoded by the given slice.
- The region of interest within a frame that is encoded by the given slice.
- The colour components encoded by the given slice.
- Any other quality parameter specified for that slice.
- Other ordering criteria may also be applied at this point.

The slices are ordered according to the score for each slice. Finally, the slices are ordered to ensure that the parent of a slice always appears before the slice itself, to ensure that every slice can be decoded as soon as it arrives.

**Delivering slices for enhancement**

Slices for enhancement are delivered using all the surplus capacity available on the communications channel. Unlike the playback case, there is no pacing or streaming of the media data. Thus the full capacity of the communications channel is used at all times even when it is not required for immediate playback of media.

**Extension to other uses**

In the description above the resulting video frames were displayed on the client computer. This technique can also be used to provide media data for other uses. For example, the frames may be used as input to an image analysis system to provide features such as shot-change detection and object recognition. The frames may also be re-compressed into a new format or stored on medium such as paper, film, magnetic tape, disc, CD-ROM or other digital storage media.

**Extension to other data types**

The technique can be applied to other data types such as audio, graphics and animation. The technique can support any data that is compressed in a format that allows enhancement as described above.

## P. Appendix 1

Some Definitions of Terms, as used in the context of the illustrated embodiments

### *Scale*

A function can be analysed into a set of components with different time/space/frequency content. These components are called scales, and the process of analysis into scales is called multi-scale decomposition. The analysis is performed by a waveform of limited duration and zero integral, called a wavelet. Components that are highly localised in space or time but have an ill-defined frequency spectrum are small-scale and capture fine detail. Components that are spread through space but have a precisely-defined frequency spectrum are large-scale and capture general trends.

### *Layer*

A Layer is a conceptual part of a media file that initialises or refines a single homogeneous component of that file, where that component is data representing spatial resolution, temporal resolution, sample fidelity, colour or any other quality axis. Hence, it is the set of data for one level of quality within a complete file; each layer is divided into slices.

### *Slice*

A slice is the part of a media file which encodes the data for a layer within a single context. Hence, each layer within a particular media file is itself divided into a series of slices, one for each context.

### *Base Layer*

A layer which is sent first to a client to initialise a representation of a media object at that client, to which refinement layers are added.

***Refinement Layer***

A Layer which is sent to a client subsequent to a Base Layer and which 'improves' the quality, according to some metric, of the representation of a media object at that client.

***Significance Layer***

A layer where all the refinement information refers to a particular bit-position for all the coefficients undergoing refinement.

***Scale Layer***

A layer for which all the refinement information is for a particular scale.

***Region Layer***

A layer for which all the refinement information is for a particular connected region in a space or time-varying function.

***Distortion***

The distortion of an image can be measured in terms of it's Peak Signal-to-Noise Ratio (PSNR) where PSNR = 10log (255^2/MSE) dB, and MSE is the image's Mean Squared Error.

***QualityTags***

The set of axes that are available to represent different aspects of perceived quality, for example, spatial resolution, temporal resolution, fidelity of a reconstructed pixel with respect to the original, etc.

***QualityDivisions***

The marking scheme for the QualityTags axes, for example, the spatial resolution axis may be marked with low, medium, high.

***QualityParams***

A set of classifiers that can be used to describe the contribution of an item of encoded media to the perceived quality of the reconstruction. A QualityParams classifier is defined by a (QualityTags=QualityDivisions) pair, for example, fidelity=6, or resolution=high.

***Codebook***

A table that provides an abstraction mechanism such that quality-manipulation operations can be defined on media files that are valid irrespective of their original compression format. The Codebook achieves format-independence through use of the QualityParams classification system.

***Filter***

An information structure that defines a partitioning of a layered media file into two parts; one part representing the media file at a reduced quality, and the other representing the enhancement information needed to reconstruct the original file. The simplest implementation is a bitmask where ‘zero’ and ‘one’ at bit positions *n* specifies whether a data item with a particular label (*n*) is or is not required in the lower-quality output file.

***Filter Mask***

A piece of information that is appended to a filtered file in order to inform downstream tools about the information content of the file (i.e, what has been filtered out of it). If the filter is implemented as a simple bitmask then the filter mask can simply be a copy of this filter.